Type Systems for Concurrent Programs

Naoki Kobayashi

Tokyo Institute of Technology

Type Systems for Programming Languages

- **Guarantee partial correctness of programs**
  - **fun fact (n) =**

    if n=0 then 1 else n $\times$ fact(n-1);

  val fact = fn: int $\rightarrow$ int

  Given an integer as an input, fact will return an integer as an output.

Type Systems for Programming Languages

- **Guarantee partial correctness of programs**

  – **fun fact (n) =**

  if n=0 then 1 else n $\times$ fact(n-1);

  val fact = fn: int $\rightarrow$ int

- **Help early finding of bugs**

  – **fun g(x) = fact(x+1.1);**

  TYPE ERROR: fact requires an argument of type int, but x+1.1 has type real.

Advanced Type Systems

- (almost) automatic analysis of:
  - Memory usage behavior (automatic insertion of “free” and “malloc”)
  - Exception (whether a raised exception is properly handled)
  - Resource usage (e.g. a file that has been opened is eventually closed)
- Type systems for low-level languages
- Type systems for concurrent languages

Type Systems for Concurrent Programs?

- **Traditional type systems (e.g. CML):**

```
fun f(n:int) =
    let val ch = channel()
    in  recv(ch)+n end
```

creates a new channel

waits to receive a value from ch

val f = fn: int $\rightarrow$ int

Type Systems for Concurrent Programs?

- **Expected Scenarios**
  - **fun f(n:int) =**

```
fun f(n:int) =
      let val ch = channel()
      in  recv(ch)+n end
```

Warning: there is no sender on channel ch

```
fun g(l: Lock) =
      (lock(l);
       if n=0 then 1 else (unlock(l); 2))
```

Warning: Lock l is not released in then-clause

Advanced Type Systems for Concurrent Programs

- **I/O mode** ([Pierce&Sangiorgi 93])
  - Channels are used for correct I/O modes.
- **Linearity** ([Kobayashi, Pierce & Turner 96])
  - Certain channels are used once for input and ouput
- **Race-freedom** ([Abad,Flanagan&Fruend 99, 2000] etc.)
- **Deadlock/Livelock-freedom** ([Yoshida 96; Kobayashi et al.97,98,2000; Puntigam 98] etc.)
  - Certain communications succeed eventually.

Type Systems for Concurrent Programs?

♦ **Expected Scenarios**

– **fun f(n:int) =**

```
let val ch = channel()
in  recv(ch)+n end
```

Warning: there is no sender on channel ch

– **fun g(l: Lock) =**

```
(lock(l);
 if n=0 then 1 else (unlock(l); 2))
```

Warning: Lock l is not released in then-clause

Outline

- ♦ Target Language
  - – Syntax
  - – Programming Examples
  - – Expected Properties of Programs
- ♦ Type System with Channel Usage
- ♦ More Advanced Type Systems
- ♦ Future Directions

Target Language: π-calculus [Milner et al.]

- Consists of basic concurrency primitives

parallel composition

$$\textbf{new } r \textbf{ in } \; c![1, r] \;|\; c?[x, y].y![x]$$

Channel creation

Message send

Message reception

$$\rightarrow \quad r![1]$$

Target Language: π-calculus [Milner et al.]

- Consists of basic concurrency primitives

$$\textbf{new } r \textbf{ in } \quad c![1, r] \;\; | \;\; c?[x, y].y![x]$$

- Expressive enough to model various features of modern programming languages
  - (Higher-order) Functions
  - Concurrent objects
  - Synchronization mechanisms (locks, etc.)

Target Language: π-calculus[Milner et al.]

P, Q **(Processes) ::=**

0	**(inaction)**
new x **in** P	**(channel creation)**
$x\,![\,v_1, \ldots, v_n\,]$	**(output)**
$x?[\,y_1, \ldots, y_n\,].\,P$	**(input)**
$P \mid Q$	**(parallel execution)**
if b **then** P **else** Q	**(conditional)**
$*P$	**(replication)**

$$x![v_1,\ldots,v_n] \mid x?[y_1,\ldots,y_n].Q \rightarrow [v_1/y_1,\ldots,v_n/y_n]Q$$

(c.f. $(\lambda x.M)N \rightarrow [N/x]M$ **)**

Example: Function Server

Server： $*succ?[n, r].r![n+1]$

Client： **new** r **in** $(succ![1,r] \mid r?\ [x]...)$

$$\underbrace{*succ?[n, r].r![n+1]}_{\textbf{server}} \mid \underbrace{succ![1,rep] \mid rep?[m].print![m]}_{\textbf{client}}$$

$$\rightarrow *succ?[n, r].r![n+1] \mid rep![2] \mid rep?[m].print![m]$$

$$\rightarrow *succ?[n, r].r![n+1] \mid print![2]$$

Example: Lock

- **Unlocked state = presence of a value**
  Locked state = lack of a value
  lock creation: new $lock$ **in** ($lock![\ ] \mid \ldots$)
  lock acquisition: $lock?[]....$
  lock release: $lock![]$

$$lock![\ ] \mid lock?[\ ].\langle CS1\rangle.lock![\ ] \mid lock?[\ ].\langle CS2\rangle.lock![\ ]$$

$$\rightarrow lock?[\ ].\langle CS1\rangle.lock![\ ] \mid \langle CS2\rangle.lock![\ ]$$

$$\rightarrow lock?[\ ].\langle CS1\rangle.lock![\ ] \mid lock![\ ]$$

$$\rightarrow \langle CS1\rangle.lock![\ ]$$

$$\rightarrow lock![\ ]$$

Desired Properties

- A server is always listening to clients' requests.
- A server will eventually send a reply for each request.

  $*\mathit{ping}?[r]$.if b then ✓ $r![1]$ else $r![2]$

  $*\mathit{ping}?[r]$.if b then ✗ 0 else $r![1]$

- A process can eventually acquire a lock.
- An acquired lock will be eventually released.

Outline

- Target Language
- Type System with Channel Usage
  - Types
  - Type-checking
  - Applications to programming languages
- More Advanced Type Systems
- Future Directions

Type System with Channel Usage

- Checks *how* (input or output) and *in which order* channels are used.
  - A reply channel is used once for output: $*ping?[r].(.....\ r![1])$
  - A lock channel is used for output after it is used for input: $lock?[].(.....\ lock![])$
- Related type systems:
  - Input/Output Channel Types [Pierce & Sangiorgi 93]
  - Linear Channel Types [Kobayashi,Pierce & Turner 96]
  - Type systems for deadlock/Livelock-freedom [Kobayashi et al]
  - Types as abstract processes [Igarashi&Kobayashi] [Rehof et al]

Channel Types

τ chan — **the type of a channel used for sending / receiving a value of type τ**

$*ping?[r:$ **int chan** $].r![\text{"abc"}]$ ✗

$*ping?[r:$ **int chan** $].r![1]$ ✓

$*ping?[r:$ **int chan** $].$**if** b **then 0 else** $r![1]$ ✓

Channel Types with Usage

τ **chan**(U) **the type of a channel used for sending / receiving a value of type** τ ***according to usage*** U

$*ping?[r\colon \textbf{int chan}(!)\,].r![\text{“abc”}]$

$*ping?[r\colon \textbf{int chan}(!)\,].r![1]$

$*ping?[r\colon \textbf{int chan}(!)\,].\textbf{if } b \textbf{ then } \mathbf{0} \textbf{ else } r![1]$

Should be used once for output

Channel Usage

U ::= **0**	not used
$?.U$	used for input, and then as U
$!.U$	used for output, and then as U
$U_1 \mid U_2$	used as U_1 and U_2 in parallel
U_1 & U_2	used as U_1 or U_2
$\mu\alpha.U$	recursion
$*U$	used as U arbitrarily many times (abbreviates $\mu\alpha.((U \mid \alpha) \,\&\, \mathbf{0})$)

Channel Usage: Example

- **Server-client connection:**

$$\mu\alpha.(?.\,\alpha) \mid *!$$

Server must be always listening to requests

Client can send requests arbitrarily many times

- **Reply channel:**

$$! \mid ?$$

- **Lock channel:**

$$! \mid *?.!$$

Lock is released first

Lock should be released each time it is acquired

Example: Lock

Should be used as a lock channel

$$newLock?[lock\!: \textbf{unit chan}(*?.!)\,].lock?[\;].\langle CS\rangle.lock![\;]$$ ✓

$$\begin{aligned}&newLock?[lock\!: \textbf{unit chan}(*?.!)\,].\\&\qquad lock?[\;].\langle CS\rangle.\textbf{if } b \textbf{ then } \mathbf{0} \textbf{ else } lock![\;]\end{aligned}$$ ✗

$$\begin{aligned}&newLock?[lock\!: \textbf{unit chan}(*?.!)\,].\\&\qquad lock?[\;].\langle CS\rangle.(lock![\;] \mid lock![\;])\end{aligned}$$ ✗

Outline

- Target Language
- Type System with Channel Usage
  - Types
  - Type-checking
  - Applications to programming langauges
- More Advanced Type Systems
- Conclusion

Type Judgment

$x_1 \colon \tau_1, \ldots, x_n \colon \tau_n \vdash P$

P is a well-typed process under the assumption that each x_i has type τ_i

Example:

- ✓ x: int chan(!) $\vdash x![1]$
- ✗ x: int chan(!), b:bool $\vdash$ if b then $x![1]$ else 0
- ✓ $ping$: (int chan(!)) chan(?) $\vdash ping?[r].r![1]$

Typing Rules

$$\frac{\Gamma,\ y{:}\ \tau,\ \ x{:}(\tau\ \textbf{chan}(U)) \quad \vdash P}{\Gamma,\ \ x\ {:}(\tau\ \textbf{chan}(?.U\ ))\vdash x?\ [y].P}$$

$$\frac{\Gamma \vdash P \qquad\qquad \Delta \vdash Q}{\Gamma\ \boldsymbol{|}\ \Delta \vdash P \mid Q}$$

Example of Type Derivation

$$\dfrac{\dfrac{r : \textbf{int chan}(!) \;\; \vdash r![1]}{ping : (\textbf{int chan}(!))\ \textbf{chan}(?) \vdash ping?[r].\ r![1]}}{ping : (\textbf{int chan}(!))\ \textbf{chan}(\omega ?) \vdash *ping?[r].\ r![1]}$$

Example of Type Derivation

$$\frac{\dfrac{r : \text{int chan}(!) \vdash r![1] \qquad r : \text{int chan}(0) \vdash 0}{r : \text{int chan}(!\&0) \vdash \text{if } b \text{ then } r![1] \text{ else } 0}}{ping : (\text{int chan}(!\&0))\ \text{chan}(?)) \vdash ping?[r].\ \text{if } b \text{ then } r![1] \text{ else } 0}$$

Outline

- Target Language
- Type System with Channel Usage
  - Types
  - Type-checking
  - Applications to programming languages
- More Advanced Type Systems
- Future Directions

Applications

- type ‘a rep_chan = ‘a chan(!);

type constructor rep_chan defined

- proc ping[r: int rep_chan] = r![1];

Process ping : int rep_chan->pr defined

- proc ping2[r: int rep_chan] =
if b then 0 else r![1] ;

Type error: r must have type int rep_chan,

but it has type int chan(0&!) in:

if b then 0 else r![1]

Applications

- type Lock = unit chan(∗?.!);

type constructor Lock defined

- proc cr[lock:Lock] = lock?[].doCR![].lock![];

Process cr: Lock -> pr defined

- proc cr2[lock:Lock] =
  lock?[].doCR![].(lock![] | lock![]);

Type error: lock must have type Lock,

but it has type unit chan(?.(!|!)) in:

lock?[].doCR![].(lock![] | lock![]);

Outline

- Target Language
- Type System with Channel Usage
- More Advanced Type Systems
  - Deadlock-freedom
  - Race analysis
- Future Directions

More Advanced Type Systems

- Type systems for deadlock/livelock-freedom [Kobayashi et al. 1997-2000]
  - A server will eventually send a reply.
    - $*ping?[r\text{: int chan}(!)].$
      $\text{new } x, y \text{ in } (x?[\,].y![\,] \mid y?[\,].(x![\,] \mid r![\,]))$.
  - A process can eventually acquire a lock, and will release it afterwards.
- Type systems for race analysis
  [Abadi, Flanagan, Freund 1999,2000]

Outline

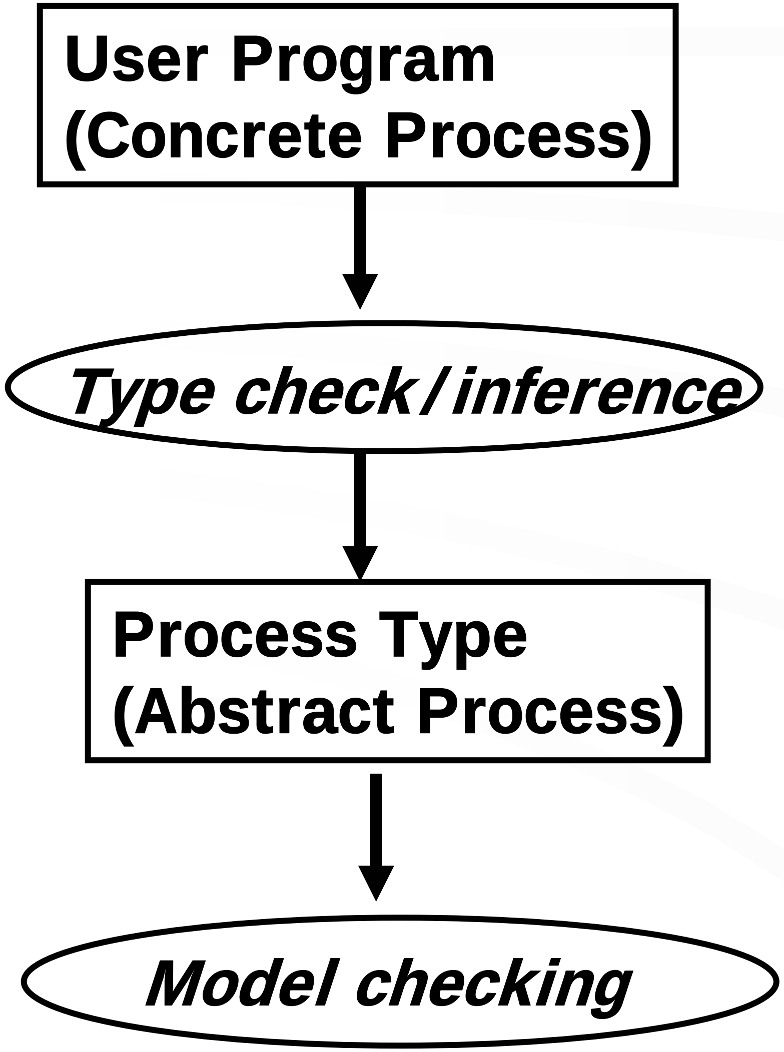
- Target Language
- Type System with Channel Usage
- More Advanced Type Systems
  - Deadlock-freedom
  - Race analysis
- Future Directions

Combination with Model Checking

- Type systems
  - Work for very large programs with infinite states
  - Properties checked are limited
- Model checking
  - Various properties can be checked
  - Work for finite state systems
  - Proper abstractions are necessary to deal with large or infinite state systems

Combination with Model Checking

[Igarashi&Kobayashi 2001]
[Chaki, Rajamani,&Rehof 2002]



$$P \overset{?}{\models} \varphi$$

$$\Downarrow$$

$$\Gamma \overset{?}{\models} \varphi'$$

Combination with Theorem Provers

ML	$\rightarrow$	**Coq, ...**
$\downarrow$		$\downarrow$
Typed π	$\rightarrow$	**?**
Fully automated, but less general		**Less automated, but more general**

Applications to Other Problems

- Analysis of Security Protocols
  - Authenticity by Typing [Gordon&Jeffrey 2001]
- Resource Usage Analysis [Igarashi&Kobayashi 2002]
  - An opened file should be eventually closed, and should not be accessed afterwards.

    File($*$(read&write); close)
  - An empty stack should not be poped.

    Stack($*$(push;(pop&0)))

Conclusion

- Type systems for concurrent programs are mature enough to be applied to concurrent languages (e.g. Race analyzer for Java [Flanagan and Freund PLDI2000])
- Future directions
  - Combination with other methods for program verification/analysis
  - Technology shift to other problems (security protocols, resource usage)